CONF-780816--4

LA-UR-78-1427

TITLE: AN IMPLEMENTATION OF THE ACM/SIGGRAPH PROPOSED GRAPHICS STANDARD IN A MULTISYSTEM ENVIRONMENT

AUTHOR(S): Richard G. Kellner
Theodore N. Reed
Ann V. Solem

SUBMITTED TO: SIGGRAPH '78

MASTER

los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87544

Form No. 836
St. No. 2629
1/75

UNITED STATES
ENERGY RESEARCH AND
DEVELOPMENT ADMINISTRATION
CONTRACT W-7405-ENG. 36

14

AN IMPLEMENTATION OF THE ACM/SIGGRAPH PROPOSED GRAPHICS STANDARD IN A MULTISYSTEM ENVIRONMENT

Richard G. Kellner
Theodore N. Reed
Ann V. Solem
Los Alamos Scientific Laboratory
P.O. Box 1663, MS-272
Los Alamos, New Mexico 87545

ABSTRACT

Los Alamos Scientific Laboratory (LASL) has implemented a graphics system designed to support one user interface for all graphics devices in all operating environments at LASL. The Common Graphics System (CGS) will support Level One of the graphics standard proposed by the ACM/SIGGRAPH Graphic Standards Planning Committee.

CGS is available in six operating environments of two different word lengths and supports four types of graphics devices. It can generate a pseudodevice file that may be postprocessed and edited for a particular graphics device, or it can generate device-specific graphics output directly. Program overlaying and dynamic buffer sharing are also supported.

CGS is structured to isolate operating system dependencies and graphics device dependencies. It is written in the RATFOR (RATional FORtran) language, which supports control flow statements and macro expansion. CGS is maintained as a single source program from which each version can be extracted automatically.

Key Words and Phrases: portability, standards, computer graphics, device-independent graphics, pseudodevice.

CR Categories: 4.22, 8.2

1. INTRODUCTION

Since 1957, the Los Alamos Scientific Laboratory (LASL) has been involved in computer graphics, pioneering work in color computer output microfilm and other areas. Currently, LASL has nine large-scale computers, several hundred Tektronix 4010-series graphics terminals, four computer output microfilm recorders, an Evans and Sutherland Picture System 2 graphics system, and numerous small computers and other graphics equipment. The computers are run 24 hours per day, 7 days per week, with a mixture of jobs, including ones that may consume all the available resources of a large computer for hours.

The gradual development of graphics software at LASL has resulted in a large collection of different capabilities and different user interfaces for each operating system and graphics device. Software maintenance, application program maintenance and conversion, and user education have required extensive effort that could be reduced by one unified graphics system. Because existing systems, such as GINO-F [4] and GPGS [3], were not suitable for our applications, we designed and implemented our own.

Late in 1976, we began implementation of the Common Graphics System (CGS). By late 1977, CGS was operational in six operating environments, on computers of two different word lengths, and was supporting four types of graphics devices. Each operating environment differs in computer hardware, operating systems, and compiler subsystems, yet the graphics interface provided by CGS is identical.

2. CAPABILITIES

CGS was patterned after the ACM/SIGGRAPH proposed standard for computer graphics software [1]. The proposed standard has four levels of capability: basic, buffered, interactive, and complete. The initial goal of CGS was to implement the basic level for the graphics devices at LASL. Subsequent versions are planned to support the buffered and interactive levels, with a long-term objective of supporting all four levels.

Included in the basic level of support are four classes of functional capability: output primitives and primitive attributes; viewing transformations for both two and three dimensions; control functions necessary to use the system; and non-retained segments. The subsequent levels of support will include: retained segments; dynamic segment attributes; input primitives; and image transforms.

Additional capabilities are necessary to support the more important LASL application programs. These are nongraphic control capabilities that must be provided as part of the graphics system. These are described below. The appendix contains a list of these control functions and the proposed SIGGRAPH control functions with which they interact.

2.1 Program Overlaying

To allow application programs to overlay CGS

routines with the application program, we have provided two control functions.

a. A function to force global variables to be allocated in the non-overlayed portion of memory.

b. A function to establish or sever linkage to a graphics device driver residing in an overlay. With our implementation of CGS and our operating environments, it is necessary for the application program to establish linkage to a graphics device driver upon entering an overlay and to sever linkage upon exiting the overlay. No view surface may be selected unless it is associated with a graphics device that is linked.

2.2 Dynamic Buffer Allocation

Application programs must be able to allocate graphics buffer space dynamically to make the memory space available to the program when it is not being used for graphics. Two functions have been provided for this purpose.

a. A function to assign a buffer for use by a view surface. The buffer is unavailable for use by the application program when it is assigned to the view surface.

b. A function to unassign a buffer. This forces the contents of the buffer to be written to disk or to an on-line graphics device. The view surface cannot be selected when a buffer is unassigned. The buffer becomes available for use by the application program.

2.3 Additional Goals

In addition to the above capabilities, the following were goals guiding the design and implementation of CGS.

a. To provide an identical interface to each of the graphics device drivers.

b. To load only the graphics device drivers requested by the application program.

c. To maintain the graphics routines and all the graphics device drivers in the same object library.

d. To support a pseudodevice (i.e., a device-independent graphics file) [2] that could be postprocessed to any graphics device. This allows graphics to be previewed and edited before specifying a graphics device for final output.

3. STRUCTURE

The structure of CGS (Fig. 1) was affected by the above functional requirements and design goals.

3.1 Device-independent Graphics

This module supports the functions described in Level One of the SIGGRAPH proposal [1]. These routines generate graphics commands in a common format for all the graphics device drivers, including the pseudodevice, and interface directly to the device selection control module. The world coordinates are processed by the viewing transform into normalized device coordinates in the range of the current normalized device coordinate space. They are then directed to the device selection control module.

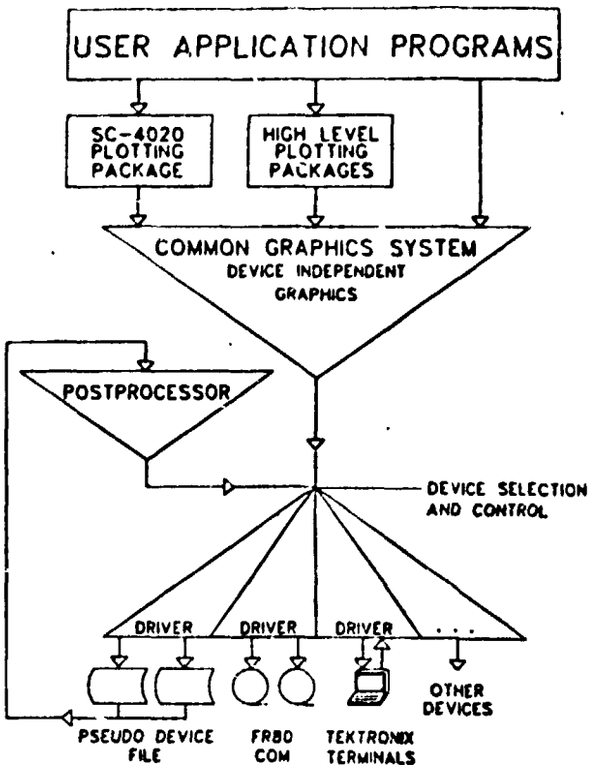

Fig. 1. Structure of CGS.

3.2 Multiple View Surfaces

CGS supports multiple view surfaces for multiple devices, including multiple view surfaces for a single device, as provided for in the SIGGRAPH proposal. Other graphics packages at LASL have allowed applications to generate a movie and slides for the same device at the same time. We have generalized this capability to generate independent streams of graphics for various graphic devices, just as a program can have any number of independent input or output files for nongraphics devices.

3.3 Device Driver Linkage

All the graphics primitive routines call a single module, the device selection control module, and pass the graphics command in a common format similar to the pseudodevice format [2]. This module calls the linked device drivers, passing the pseudodevice command along. The application program must name all the device drivers that may be needed, and only those will be loaded. This is necessary because all of our loaders are static overlay loaders. The drivers may all reside in memory at the same time, or the applica-

tion program may overlay drivers with other routines. Drivers not in memory must be unlinked; drivers in memory may be linked or unlinked. Since all device drivers may be in memory simultaneously, all device driver routines have unique names.

3.4 Graphics Device Drivers

Each device driver generates device-specific information. It transforms the normalized device coordinates to the coordinate space of the graphics device. Each device driver checks for all view surfaces that are assigned to it and sends the plotting information to each view surface that is currently selected. This is accomplished by maintaining a separate workspace containing variables and working storage for each view surface. Thus, "output-only" device drivers write to a separate file on disk for each selected view surface. This capability could also be used to drive several terminals of the same type if the operating environment can support multiple terminals per job.

4. IMPLEMENTATION

Important implementation considerations were: portability; maintainability; compatibility with an existing plotting package; and a modular implementation.

4.1 Portability

The biggest problem in portability was caused by the lack of any standards in system interfaces. Capabilities such as logical and shifting operations, bit and byte packing, character set conversion, in-core formatted data conversion, and job and system inquiries, are system functions that are sufficiently well understood to warrant some standardization across operating environments. We developed one system-independent interface to the operating environments. To move to a new operating environment, only the routines defining that interface need be rewritten.

In addition to the system functions mentioned above, we defined a system-independent I/O interface for creating a file; opening and closing a path from the program to a file or to an on-line graphics device; disposing a file to its ultimate destination, such as an off-line graphics device; and reading and writing any number of bits.

At LASL we are required to generate a graphics file in one operating environment and read it in another operating environment. In order to eliminate conversions between these environments, all graphics files have the same format. Each file has an index indicating where physical record marks are to be inserted, because some graphics devices require a particular structure to the physical records or to an input graphics tape. The index also records frame addresses to facilitate editing and postprocessing.

4.2 Maintainability

CGS exists as one master source program, from which a version for each operating environment can be extracted. This ensures uniformity in the graphics system in all operating environments.

Two types of preprocessing are used to extract each version from the master source.

a. All system-dependent and device-dependent quantities, such as computer word size, graphics device command size, and output buffer size, are set with macros. Preprocessing with the right set of macro definitions causes all the quantities to be applied throughout the source program.

b. The modules that define the system-independent interface must be different for each operating environment. Those modules that have been optimized with in-line logical and shift operations also have to be different. This conditional compilation is achieved by flagging each system-dependent statement with a system ID, and preprocessing to delete all lines that do not belong to the desired version.

We use the RATional FORtran (RATFOR) language from Bell Laboratories [5,6]. The RATFOR preprocessor outputs ANSI standard FORTRAN. It supports control flow statements, such as IF/ELSE and WHILE, and also has a limited macro capability.

The project librarian generates the graphics system for all operating environments--compiling; building libraries and utilities; testing; saving all listings on microfiche; keeping records; releasing the new libraries and utilities for public use; and keeping backup copies. The librarian uses procedure files and macros so that each major step can be easily performed.

4.3 SC-4020 Emulation

There is a tremendous investment at LASL in application programs that produce graphics output for the SC-4020 microfilm recorder. Conversion of these programs to run under CGS had to be minimized, so we rewrote the SC-4020 primitive routines to call the CGS primitives. Existing high-level routines and application programs have gained portability across operating environments and graphics devices with almost no conversion of the graphics portions of their codes.

4.4 Modular Implementation

Our software implementation called for installing a minimum version of CGS, suitable for use by production programs in all operating environments. The first release of CGS was a two-dimensional system, which supported only the pseudodevice file, and included the SC-4020 emulation routines. A postprocessor supporting several of our graphics devices was written utilizing existing graphics software. Once operational, CGS was optimized and enhanced uniformly in all operating environments and implementation of device drivers suitable for use directly with CGS was begun.

5. CONCLUSIONS

Our initial objective was to implement device-independent graphics in a portable fashion to support existing application programs in new

and existing operating environments at LASL. The conversion of existing high-level graphics routines to use CGS brought widespread availability quickly and gave many existing application programs access to a variety of graphics devices and operating environments. The resulting user feedback has been valuable, especially in the areas of portable software and ascertaining user needs for improved high-level graphics.

Writing the system with portability in mind and then looking at efficiency has resulted in several modifications to our portability techniques, primarily to improve efficiency.

Level One of the SIGGRAPH proposal has been sufficient to support the majority of our existing applications. The additional device driver control functions we have provided have enabled CGS to support several of our larger, more important applications. It will soon be necessary to support a Level Three implementation to provide interactive graphics using CGS.

As of early 1978, four and one-half man-years have been expended in the development, documentation, and maintenance of a partial Level One implementation of the SIGGRAPH proposal in six operating environments. This also includes the conversion of 9000 lines of high-level graphics routines and many hours of user education and consulting.

6. APPENDIX

6.1 Initialization and Termination

6.1.1 ALLOCATE-GLOBALS ( )

This function allocates CGS global variables. The allocation occurs wherever the routine is loaded. This function may be called anytime.

Errors: None.

6.1.2 INITIALIZE-CORE (LEVEL)

This function initializes the core. Except for ALLOCATE-GLOBALS, it must be the first call made.

Errors:
1. CGS was already initialized.
2. The specified level is not supported.

6.1.3 TERMINATE-CORE ( )

This function releases all resources being used by CGS.

Errors:
1. CGS was not initialized.
2. A view surface has not been terminated.
3. A view surface has not been closed.

6.2 Device Selection and Control

6.2.1 OPEN-VIEW-SURFACE (SURFACE-NAME, BUFFER-SIZE)

This function associates a buffer with the specified view surface.

Errors:
1. The view surface was already open.

6.2.2 CLOSE-VIEW-SURFACE (SURFACE-NAME)

This function disassociates a buffer from the specified view surface.

Errors:
1. The view surface was not opened.
2. The view surface was not deselected.

6.2.3 LINK-DEVICE-DRIVER (DEVICE-NAME, DRIVER-ADDRESS)

This function generates the link to the specified device driver in the device selection module.

Errors:
1. The device driver was already linked.
2. Unrecognized device name.

6.2.4 UNLINK-DEVICE-DRIVER (DEVICE-NAME)

This function eliminates the link to the specified device driver in the device selection module.

Errors:
1. The device driver was not linked.
2. Unrecognized device name.
3. A view surface assigned to the device was not deselected.

6.2.5 INITIALIZE-VIEW-SURFACE (SURFACE-NAME, DEVICE-NAME, DEVICE-OPTIONS)

This function associates a view surface with a device driver. The device options are used to select specific device capabilities. These options are used to initialize the view surface.

Errors:
1. The view surface was already initialized.
2. Unrecognized device name.
3. The view surface was not opened.
4. The associated device driver was not linked.

6.2.6 TERMINATE-VIEW-SURFACE (SURFACE-NAME)

This function terminates the specified view surface.

Errors:
1. The view surface was not initialized.
2. The view surface was not deselected.
3. The view surface was not opened.
4. The associated device driver was not linked.

6.2.7 SELECT-VIEW-SURFACE (SURFACE-NAME)

This function selects the specified view surface for subsequent graphic output.

Errors:
1. The view surface was already selected.
2. The view surface was not initialized.
3. The view surface was not opened.
4. The associated device driver was not linked.
5. A segment is open.

6.2.8 DESELECT-VIEW-SURFACE (SURFACE-NAME)

This function deselects the view surface.

Errors:
1. The view surface was not selected.
2. A segment is open.

ACKNOWLEDGMENTS

We would like to acknowledge Raymond Elliott of LASL for the support and encouragement he gave this project, as well as his many comments and suggestions. We would also like to thank Jeanne Hurford of LASL for word processing support.

We also acknowledge the joint effort concerning graphics standardization between Sandia Laboratories at Albuquerque, the Air Force Weapons Laboratory at Kirtland Air Force Base, and LASL.

We are especially indebted to the many members of the ACM/SIGGRAPH Graphic Standards Planning Committee. Their report sets forth the first real hope that a standard for computer graphics software will be established.

REFERENCES

1. ACM. Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH. Computer Graphics 11, 3, Fall 1977.

2. Conley, B., et al. Basic Graphics Package Intermediate File Format. LASL internal document, March 1978.

3. General Purpose Graphics System. Katholieke Universiteit, Nijmegin, The Netherlands, 1975.

4. GINO-F User Manual. Computer Aided Design Centre, Cambridge, England, 1974.

5. Kernighan, B.W. RATFOR - A Preprocessor for a Rational FORTRAN. Bell Laboratories, Murray Hill, NJ.

6. Kernighan, B.W., and Plauger, P.J. Software Tools. Addison-Wesley, Reading, MA, 1976.